Intelligent Systems for Autonomous Aircraft

K. Krishnamurthy and D. T. Ward
Texas A&M University
College Station, TX 77843

Abstract

This paper introduces a mode-driven intelligent agent for the control of autonomous aircraft. The agent architecture uses pilot-type classifications of aircraft modes within all decision-making and reasoning processes. An essential component of the intelligent agent is a neuro-fuzzy inference engine that infers the current mode from sensor data and provides abstracted input for decision-making. The development of the intelligent system is structured around a nonlinear simulation of a conceptual V/STOL Uninhabited Aerial Vehicle. We are incrementally adding capabilities to achieve a multi-faceted surveillance and combat-capable intelligent pilot-substitute.

1 Introduction

The military use of unmanned aircraft has increased significantly in the recent past [1]. These aircraft isolate human operators from risk and thereby offer tactical benefits and reduced operating costs to the armed forces. Current research to increase the utility of these aircraft has focused, perhaps inevitably, on enhancing their autonomy. An USAF document on future technologies [6] foresees the widespread use of self-piloted aircraft for reconnaissance and even combat, and predicts that the remote operators of such aircraft "will provide general direction in real-time when necessary."

The synthesis of an airborne intelligent control system to interpret and implement such "general directions" is a prodigious task. Aircraft are highly nonlinear under-actuated dynamic systems, and differ sharply from traditional mobile robots in their inability to "stop and think." Autonomous aircraft – also called Uninhabited Aerial Vehicles, or UAVs – must perform automated planning, scheduling and self-monitoring, incorporate low-level control strategies, and coexist with manned aircraft in a mixed theater of operations. In order to satisfy these demands, the onboard intelligent system must utilize a judicious blend of aviation practice, control theory and AI concepts.

In this paper, we propose a pilot-like intelligent agent that demonstrates this blend of techniques. This agent fits into a classical hierarchical framework for intelligent control, depicted in Figure 1. We view the knowledge element of this framework to be a replacement for a pilot's knowledge of aircraft operations and UAV capabilities, while the control element is a set of advanced low-level control techniques. The reasoning element in this framework is the Intelligent Flight Director (IFD), which is the focus of our work.

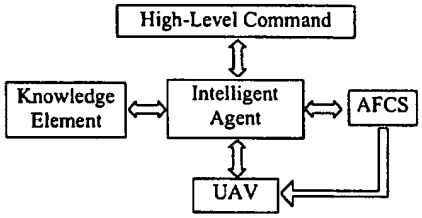

Figure 1: Hierarchical framework for intelligent control

2 Intelligent Flight Director

Our design for the IFD relies heavily on the reasoning and inference methods used by pilots to fly missions. In general, pilots implement commanded missions as sequences of maneuvers (well-defined mission segments characterized by patterns in sensor data). Pilots also use abstracted knowledge of the current maneuver to interpret a high-level command, which is often terse and incomplete. For instance, while aircraft possess 6 degrees of freedom, the commands received by pilots are commonly in terms of only four dynamic states (see Table 1). This interpret-and-implement process enables the pilot to overcome the minimal information provided, while exploiting standard aviation practice and knowledge of aircraft operations. This process also provides intuitive means to apply aircraft constraints and use aircraft capabilities appropriately.

Table 1: Pilot-level and aircraft-level states

Pilot-level states	Position, Airspeed, Altitude, Heading
Aircraft-level states	3 linear velocities and accelerations, 3 inertial angles, rates and accelerations

Analogous to the above outline of human pilot operations, the IFD implements commands as a series of maneuvers through an interpret-and-implement cycle. The resulting internal architecture of the IFD is presented in Figure 2 and features four sub-agents. Each of these agents has a specific task, as suggested below:

- *Command Logic*: Translation of the high-level commands into a sequence of maneuvers
- *Trajectory Comparator*: Evolution of trajectories for each of the sequenced maneuvers
- *Switching Logic*: Selection of a strategy to track the state trajectories most important for each maneuver
- *Mission Segment Identifier*: Inference of the current maneuver from sensory inputs.

0-7803-6583-6/00/$10.00 © 2000 IEEE

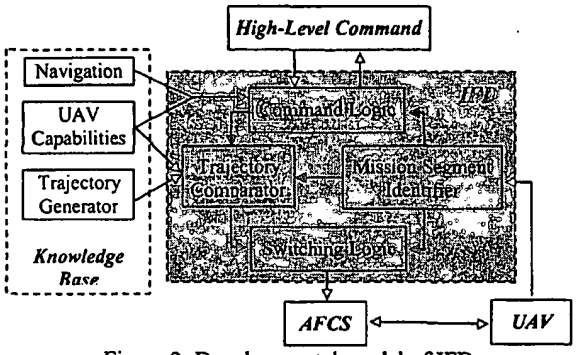

Figure 2: Developmental model of IFD

The Command Logic, Trajectory Comparator and Switching Logic are decision-making systems; the Mission Segment Identifier (MSI) is an inference system. In general, this architecture should also include modules to execute health-monitoring and system identification tasks. However, these detailed topics of research are beyond the scope of our immediate interest, and are not addressed as part of the initial development of the IFD.

The cooperative nature of the intelligent system is indicated by the high inter-connectivity of the four modules in the IFD, and that between the IFD and the Knowledge Base. In the subsequent parts of this paper, we discuss the development of each module in the IFD and provide sample operations using a nonlinear batch simulation of a conceptual V/STOL UAV.

3 Command Logic

This sub-agent interprets high level commands to ensure that all target states are known to the system, and generates a sequence of maneuvers to attain target states. Methods to perform these two tasks are discussed separately below.

3.1 Command interpretation

As stated earlier, the high-level command is terse and sometimes incomplete. Before maneuvers can be sequenced to implement the command, the IFD must infer the nature and content of the command, and any missing pilot-level states must be inferred using a logical process.

In our conceptual system, there exist two types of commands, classified according to the absence or presence of a significant cruise segment. The first type of command involves local maneuvering only (for example, a holding pattern) – such commands can be decomposed into maneuvers directly. In contrast, the second type of command involves a significant cruise segment (for instance a flight between two waypoints in a flight plan). These commands require an extra decomposition step – the insertion of appropriate intermediate goal states. This additional step ensures that the overall command is implemented as three commands – one, an initial command to initiate the cruise segment; two, a cruise of the required duration; and three, a final command to attain the final states. Each of these three sub-commands can then be treated as a command of the first type.

For both these types of commands, the Command Logic module must determine values for states that are not specified by the high-level command. The logical process that achieves this result is presented as a decision tree in Table 2 below. Whenever necessary, the IFD refers to the Knowledge Base to select values compatible with the mission, the mission segment and aircraft capabilities. This interpretation process results in a complete list of initial and target pilot-level states and mission segments for the overall command. The Command Logic now possesses sufficient information to sequence maneuvers.

Table 2: Logical process for command interpretation

Quantity	*Specified?*	*Inference*
Position	Y	Flight is Manv – S&L – Manv
	N	Flight is Local Manv
Heading	Y	Target Mission Segment = S&L
	N	Target Mission Segment = Holding
Altitude	Y	—
	N	Choose value
Airspeed	Y	—
	N	Choose value
Target mission segment	Y	Overrides above inferences
	N	As inferred above; else Holding

3.2 Maneuver Sequencing

Pilots sequence maneuvers through a logical process, aided by knowledge of aircraft operating procedures. In the IFD, maneuver sequencing is performed by a simple rule-base, supplemented by logical checks for constraint violations. The algorithm for this process is depicted in Figure 3.

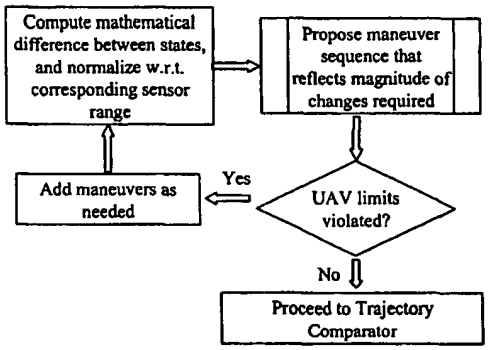

Figure 3: Algorithm for maneuver sequencing

The algorithm operates as follows. The rule-base uses the differences between the initial and final states to select appropriate maneuvers, which are then sequenced in terms of relative magnitudes. However, this process does not consider aircraft performance limits; hence, the proposed maneuver sequence is checked (using the Knowledge Base) for violations of aircraft limits. Specifically, the present values of the states are inspected for compatibility with the maneuver sequence. If necessary, additional maneuvers are added at the beginning of the proposed sequence. This process culminates in a feasible maneuver sequence.

To illustrate these concepts of interpretation and sequencing, Table 3 presents schematically the response of the Command Logic module to a high-level command to proceed to a specified waypoint.

Table 3: Sample operation of Command Logic

Command:	Proceed to waypoint X
Current States:	
▫ Position	Obtained from Nav sensors
▫ Airspeed	20 fps
▫ Altitude	50 feet
▫ Mission Segment	"Hover"
Command Logic:	
▫ Interpretation	Calculate heading to waypoint – extract cruise altitude & A/S
▫ Sequencing	Results in "accel – turn – climb"
▫ Check limits	No violations
▫ Add maneuver	None required
▫ Final sequence	"accel – turn – climb – hold"

We assume that the UAV is initially at hover. Since a target position is specified, the Command Logic infers that a significant cruise segment is required. Cruise parameters are extracted from the Knowledge Base, and the direction to fly to the waypoint is calculated. These values are then set as intermediate goals. The algorithm of Figure 3 is used to select maneuvers, resulting in the sequence indicated in Table 3.

Each of these simple maneuvers will achieve a few of the desired changes between initial and target states. Since a target state is not known explicitly, the UAV will enter a holding pattern at the destination. Of course, in actual operations the Command Logic provides numerical values for all states – as will be indicated in the subsequent discussion of the Trajectory Comparator.

The present system can provide feasible maneuver sequences for representative high-level commands. However, it cannot fully exploit UAV performance and control capabilities by superposing operations (to produce climbing turns, for example). In order to overcome this limitation, we must develop methods to measure the available spare capabilities, perhaps using aircraft agility metrics [10]. This task has not yet been attempted.

4 Trajectory Comparator

This module cooperates with the Knowledge Base to provide suitable trajectories for the maneuvers sequenced by Command Logic. The process infers aircraft-level states from pilot-level states, an increase in complexity tackled partly within the Trajectory Comparator, and partly by a Trajectory Generator [11] within the Knowledge Base.

Within the Trajectory Comparator, we have encoded qualitative pilot knowledge of the behavior of states and en-route limits during each maneuver. This encoding permits this module to specify aircraft attitudes, airspeed and altitude for each maneuver. An important feature of this module is that some maneuvers are sub-divided into three "maneuver segments" – a short-duration "initiation," a long-duration "body," and a short duration "conclusion." Sub-dividing a maneuver into component segments permits better representation of pilot knowledge and closer control of applicable constraints.

Within the Knowledge Base, the Trajectory Generator uses pilot knowledge, stored as prototype maneuvers, to interpret the qualitative information provided by the Trajectory Comparator. This trajectory generation algorithm uses the prototype maneuvers, aircraft stability derivatives and aircraft equations of motion to synthesize sub-optimal polynomial state trajectories.

Table 4: Example of Trajectory Comparator operation

Command: Proceed to waypoint X

Sequence: Level Accel (LA) – Steady Turn – Climb

		LA	Steady Turn			Climb
			Init.	Body	Conc.	
Initial States	AS	20	400	400	400	400
	Alt	50	50	50	50	50
	ϕ	0	0	60	60	0
	θ	n.s.	n.s.	n.s.	n.s.	n.s.
	φ	n.s.	0	n.s	n.s.	45
Final States	AS	400	400	400	400	400
	Alt	50	50	50	50	5000
	ϕ	0	60	60	0	0
	θ	n.s.	n.s.	n.s.	n.s.	n.s.
	φ	n.s.	n.s.	n.s.	45	45

We illustrate this concept with an example in Table 4, which depicts the operation of the Trajectory Comparator in response to the command discussed earlier (Table 3): "Fly to waypoint X." The Command Logic decomposes this command into three maneuvers: level acceleration to a safe speed (400 fps in this example), a steady turn to the required heading (turn through 45 degrees) and a climb to cruise altitude (5000 ft.). The Trajectory Comparator divides the steady turn maneuver into maneuver segments, and assigns initial and final state values for each segment (Table 4). En-route constraints extracted from the UAV capability envelope are also specified for each segment, the

whole process forcing the creation of more feasible, pilot-like trajectories for each maneuver [11].

This example illustrates the utility of maneuver-based operations. In the level acceleration maneuver, since load factor is "known" to be unity, the trajectory generator calculates three velocity components and the pitch attitude, θ, from first principles. Further, since the trajectory generation algorithm uses prototype maneuvers, it does not require explicit values for non-critical states. For example, it is "known" that aircraft heading, ψ, continually changes throughout a turn maneuver. Hence, the Trajectory Comparator indicates only the total heading change required (by specifying heading at the beginning and at the end of the turn), leaving intermediate heading values unspecified.

This brief discussion provides an overview of the operation of this sub-agent; further details can be found in [3]. A feature that would enhance the functionality of this module is the estimation of the time spent in each maneuver segment. An initial guess for time would assist the sub-optimal trajectory generator in converging to a feasible solution. Another desirable feature would be an estimation of the "merit" of the generated trajectories, to allow tweaking of the applied constraints. These are topics to be addressed in continued development of our system.

5 Mission Segment Identifier

This sub-agent contributes situation awareness to the IFD. The development of this inference system has built upon earlier work on the inference of aircraft flight modes from sensor data [8] and has addressed several issues faced in the course of that exercise. Details of this development process and sample operations can be found in [3,4]. In the current paper, we describe the system briefly and present sample results to explain the concept.

Like the Trajectory Comparator, the MSI is built around the concept of maneuver segments – it first infers maneuver segments from sensor data, and then identifies the overall maneuver. In the resulting two-level architecture for the MSI, a neural inference system performs maneuver segment identification, and a fuzzy rule-base infers the maneuver from the sequence of maneuver segments.

A breakthrough in synthesizing training data, coupled with the existence of powerful training methods, influenced our selection of a neural network-based inference engine for maneuver segment identification. We use a symmetric feed-forward network with two identical hidden layers and trained using a back-propagation algorithm. Instead of collecting training data from multiple simulated flights, we use a-priori knowledge of the expected data patterns for each maneuver segment to synthesize heuristic training data sets. This approach has worked quite well [4].

The inputs to the inference system are raw data from the sensors onboard the UAV. These data are first normalized with respect to full-scale ranges of the appropriate sensors. These normalized data are the inputs to the neural network. Faced with ambiguous data and sensor noise, the raw outputs of the neural network sometimes exhibit 'chatter,' a rapid switching back-and-forth between identified segments. We use a mode-memory scheme (inspired by human reasoning in the presence of fluctuating data) to overcome this phenomenon. The network outputs are also normalized to ensure that processed outputs sum to unity – we refer to these normalized outputs as confidence values.

At the next level, maneuver identification is performed by a set of heuristic rules that relate each maneuver to the sequence and duration of component maneuver segments. Since ambient disturbances and system non-linearities can cause variations from the specific numerical descriptions coded in these rules, we use a fuzzy inference system built on this rule-base.

In operation, the sequence of identified maneuver segments is continually monitored to detect segment changes, and the duration of each segment is stored. By comparing the duration of successive segments, this process can reject transients – this serves as a second line of defense against chatter. The stored duration data are inputs to a Mamdani-style fuzzy inference system [5], the raw outputs from which reflect the degrees to which the rules match the inputs. Since there are often more rules than maneuvers, these outputs are grouped and normalized to obtain confidence values for each maneuver.

Table 5: Current version of the MSI

Sensed data	$\dot{u}$	Rate of change of forward velocity		
	Φ	Rate of change of bank angle		
	$\dot{\Psi}$	Rate of change of heading		
	$\dot{h}$	Rate of change of altitude		
Maneuver Segments	1	Level Acceleration	4	Climbing Turn
	2	Rolling Transient	5	Aggressive Turn
	3	Steady Turn	6	Straight and Level
Maneuvers	1	En Route Turn	4	Horizontal Break
	2	Holding Pattern	5	Cruise
	3	Vertical Break		

The current version of the MSI uses four sensor inputs to infer six maneuver segments and five maneuvers (Table 5). We illustrate the operation of this version of the MSI with a simulated holding pattern (Figure 3), a common maneuver comprised of alternating standard turns and level flight, with each of these segments lasting for a minute. The data from this benign maneuver have low signal-to-noise ratios, as can be inferred from the normalized values of $\dot{\Psi}$ and Φ in Figure 3. We have studied the performance of the MSI in the presence of sensor noise, by superposing 2.5% random zero-mean noise on these data. The results of the inference process (with superposed noise) are depicted in Figure 4.

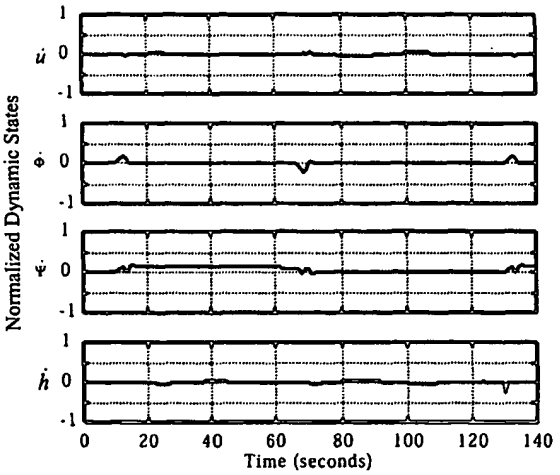

Figure 3: Simulated Holding Pattern

The first plot of Figure 4 is a time-history of the maneuver segments identified by the first level of the MSI. The numbers on the abscissa refer to the maneuver segment indices in Table 5. The second plot of Figure 4 presents the inferred maneuver, while the last plot gives the associated confidence measures. These results indicate intuitive behavior – the maneuver initially identified is an en-route turn, and this inference changes to a holding pattern as the duration of the turn approaches 60 seconds.

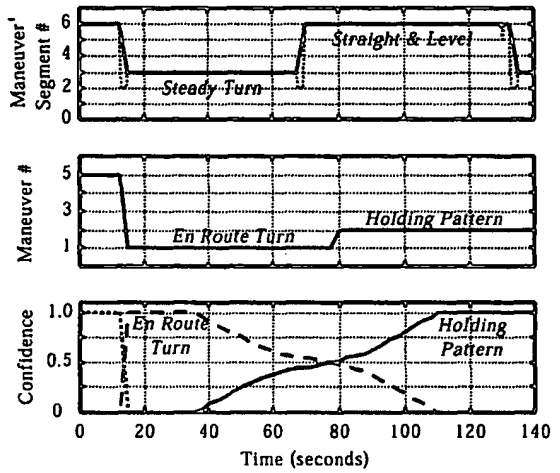

Figure 4: Maneuver identification – Holding Pattern

6 Switching Logic

This module of the IFD implements the high-level command through the Automatic Flight Control System (AFCS). In keeping with the mode-driven nature of our intelligent agent, the Switching Logic uses maneuver-based logic to select control strategies; this sub-agent also ensures that accurate mathematical models of the aircraft are available for use by the selected controller. These tasks are interrelated and are intricately coupled to the available set of control laws.

6.1 Controller selection

Aircraft are under-actuated dynamic systems, possessing four actuators with which to control motions in 6-DOF space. Further, owing to aerodynamic characteristics, aircraft equations of motion involve eight (sometimes 10) dynamic states [7]. In order to overcome this under-actuation problem with our pilot-like intelligent system, we envision the use of a family of maneuver-based controllers. Each such controller would use the four actuators to track a maneuver-specific subset of (up to four) dynamic states, while regulating undesirable perturbations in the other states. With this approach, the selection of a specific control strategy would be triggered by the current maneuver, as commanded and inferred by the other modules of the IFD.

Our current research [3] seeks to create this family of tracking controllers using optimal control theory to construct maneuver-based linear quadratic trackers (LQTs). These LQTs use state vectors augmented with tracking errors in the dynamic states of interest. By regulating these error terms to zero, we achieve the tracking goal; by regulating the other states, we minimize perturbations in those quantities. The relative importance of these tracking and regulating functions are adjusted through the weighting terms in the cost function.

To illustrate this control concept, Figure 5 presents the performance of a LQT controller designed to execute a holding pattern. This controller successfully tracks three angular rate trajectories (indicated by the low magnitudes of the first three plots of Figure 5), while simultaneously regulating the perturbations in the altitude trajectory (last plot of Figure 5).

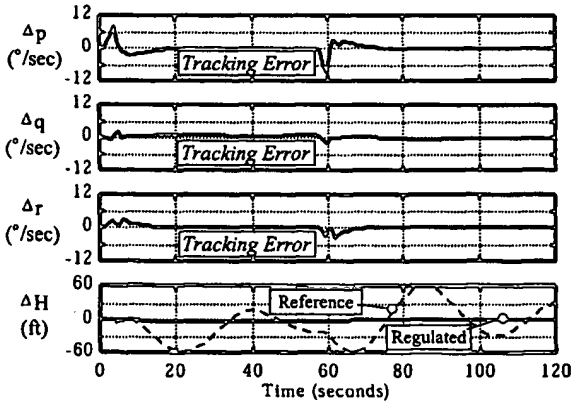

Figure 5: Sample operation of a maneuver-driven LQT

The above control concept cannot guarantee performance in the presence of actuator failures or in non-linear portions of the aircraft flight envelope. Therefore, this family of linear controllers must be augmented by other control strategies onboard the UAV. This realization motivates the development of logic that monitors the tracking errors as well as state behaviors, and triggers a switch to more robust controllers when necessary. Development of this logic must be a focus of continuing work with this concept.

6.2 Model selection

The IFD uses the Knowledge Base as a repository for all existing knowledge of the aircraft. This knowledge includes mathematical models of the aircraft – a large number of these models are stored a-priori, and constitute the primary source of models for control purposes. However, there are portions of the UAV flight envelope that cannot be well represented (high-α, for instance), and there are situations where the stored models are rendered inaccurate by damage (actuator failures, etc.). In such situations, the Switching Logic uses online system identification to obtain a more-current model.

To have this capability, the Switching Logic must continually check the validity of the model currently in use. However, the UAV is always in closed-loop control, and so model inaccuracies are compensated by the controller. The only available indicator of math model inaccuracy is the control effort required to track a given trajectory.

This observation forms the basis of the model selection logic in the IFD. Since the reference trajectories are always provided with nominal control histories, the Switching Logic continually computes the norm of the deviation in control effort. When this norm exceeds heuristic limits, this module triggers fresh system identification. The limits on the allowable control effort deviations are indicative of the confidence associated with the models as well as the requirements of the maneuver.

7 Conclusions and future work

We have outlined a mode-driven intelligent control system concept for UAVs. The modes used in this concept are maneuvers that the UAV can perform. The system decomposes and implements all commanded tasks as a sequence of these modes. It also monitors the present mode for future command execution, operational checks and internal inference processes. Since these maneuvers are intuitive to pilots, this mode-based architecture can exploit an enormous body of existing knowledge and experience with aircraft operations. This advantage is evident in the coding of logic processes, in the online generation of trajectories and in the selection of aircraft control strategies.

The pilot-like features of this concept are largely absent from past work involving AI concepts in aerospace applications [2,9]. Our development of this concept is very much an ongoing process. Successes to date lead us to believe that this concept is a promising basis for development of a practical intelligent control system for UAVs.

Future work on the IFD must concentrate on developing the Switching Logic module, by integrating the IFD with a suite of advanced control strategies. The relationship between control effort and model accuracy must also be quantified to improve the process of model selection. Increasing the number and complexity of maneuvers to be considered will be necessary for future development of all modules of the IFD.

References

[1] J. W. Canan, "Seeing more and risking less with UAVs," *Aerospace America*, October 1999, pp. 26–31.

[2] P. E. Jones, J. E. Laird, et al, "Automated Intelligent Pilots for Combat Flight Simulation," *AI Magazine*, Spring 1999, Vol. 20, No. 1, pp. 27-42.

[3] K. Krishnamurthy, "Intelligent Systems for Autonomous Aircraft," Ph.D. Dissertation, Department of Aerospace Engineering, Texas A&M University (to be completed December 2000).

[4] K. Krishnamurthy and D. T. Ward, "Automated Mode Inferencing for Intelligent Aircraft," *Proceedings of the 2000 IEEE International Conference on Control Applications* (to appear).

[5] E. H. Mamdani and S. Assilian, "An Experiment in Linguistic Synthesis with a Fuzzy Controller," *International Journal of Man-Machine Studies*, Vol. 7, No. 1, 1975, pp. 1-13.

[6] G. H. McCall (Study Director), *New World Vistas – Air and Space Power for the 21st Century, Summary Volume*, United States Air Force, Dec. 1995, pp. 28.

[7] R. C. Nelson, *Flight Stability and Automatic Control*, McGraw-Hill Inc., New York, NY, 1989, pp.83-et seq.

[8] J. H. Painter, W. E. Kelly III, et al, "Decision Support for the General Aviation Pilot," *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 1, pp. 88–93.

[9] B. Pell, D. E. Bernard, S. A. Chien, et al, "An Autonomous Spacecraft Agent Prototype," *Autonomous Robots*, March 1998, Vol. 5, No. 1, pp. 29–52.

[10] D. M. Smith and J. Valasek, "Agility Metric Robustness Using Linear Error Theory," AIAA Paper 99-4093, *Proceedings of the AIAA Atmospheric Flight Mechanics Conference*, August 1999.

[11] A. Verma, "Determination and Stable Tracking of Feasible Aircraft Trajectories Using Adaptive Inverse Dynamics," Ph.D. Dissertation, Department of Aerospace Engineering, Texas A&M University, May 2000.

VIATO - Visual Interactive Aircraft Trajectory Optimization

Kai Virtanen, Harri Ehtamo, Tuomas Raivio and Raimo P. Hämäläinen

Systems Analysis Laboratory
Helsinki University of Technology
P.O.Box 1100
FIN-02015 HUT
Finland

Abstract

An approach towards automated solution of aircraft trajectory optimization problems is introduced and applied to an interactive MS-Windows software called 'Visual Interactive Aircraft Trajectory Optimization' (VIATO). The software solves, e.g., minimum time trajectories to a fixed or to a moving target. It consists of an automated optimization routine and a graphical user interface. The software is easy-to-use and thus its user does not need to know optimal control theory and mathematical modeling. Most mathematical models of flight vehicles are structurally similar and differ only in parameters. In VIATO software the equation of motion and the state and the control constraints are fixed in advance and different aircraft types are represented as a set of parameters. Specifying also objective functions of the optimal control problems the explicit formulation of optimal control problems is totally avoided. Reliable convergence of solution methods is achieved by replacing the original infinite dimensional problem with a finite dimensional approximation. The approximation is solved using nonlinear programming.

1. Introduction

In aircraft trajectory optimization problems, flight paths that minimize the objective function of the problem subject to state and control constraints are searched. The objective can be, for example, the total flight time or fuel consumption. The optimal desing of an aircraft trajectory can be handled using optimal control theory. However, the solution of an optimal control problem is a laborious and time-consuming task. In this paper an approach towards automated solution of a fixed class of aircraft trajectory optimization problems is introduced and implemented in Visual Interactive Aircraft Trajectory Optimization (VIATO) software.

Most flight vehicles obey structurally similar mathematical models that differ only in parameters. Hence the equations of motion and the structure of possible constraints for control and state variables can be fixed in advance, and different aircraft types can be characterized by a set of parameters. The parameters include quantities like drag coefficients and thrust force but also constraint bounds like the g-limits and largest allowed dynamic pressure.

Indirect solution methods solve optimal control problems using necessary conditions for optimality arising from Pontryagin's maximum principle (see, e.g., [4]). In most cases these methods are not appropriate for automating the solution process, since their convergence domain is small and the switching structure of the solution must be known in advance. Larger convergence domain is obtained by converting the original problem into a nonlinear programming problem that is solved using methods of nonlinear optimization (see, e.g., [1, 9]). Furthermore, the derivation of the necessary conditions as well as the estimation of the switching structure are avoided. When a graphical user interface is ported to the automatic optimization routine, solutions become also accessible to users who are not expert in the area of optimal control theory and computer science. The solution method requires moderate computational effort and thus the optimization solver can be run in a PC.

Based on these considerations, we have developed VIATO software. It consists of a graphical user interface and an optimization server. The user interface operates under MS-Windows. It maintains an aircraft model library, provides the optimization server with required information and displays optimal trajectories graphically. The server solves the discretized problems using NPSOL program [8], which is a versatile implementation of sequential quadratic programming (see, e.g., [6]). When solving a particular trajectory optimization problem, the user only needs to choose an aircraft model type from the model library, choose the objective of the problem and specify initial and terminal states. Sensitivity of the optimal objective value and the optimal state and control history with respect to different model parameters can also be studied.

Some programs for solving optimal control problems are already available. The user of all following packages must formulate the optimal control problem to be solved and thus the user must be familiar with optimal control theory. The NPDOT package [10] discretizes the original optimal control problem by direct collocation and nonlinear programming is used to solve the finite-dimensional problem. VTOTS software package [2] consists of a finite-element algorithm together with a multiple shooting algorithm for solving the infinite-dimensional problem. VTOTS is not a stand-alone optimization software, because it uses three computer

0-7803-4053-1/97/$10.00 © 1997 IEEE

languages and the resulting files must be compiled. The aim of these two programs is to solve general optimal control problems. The POST [3] and the OTIS programs [10] are aimed at solving aerospace problems and produce optimal point-mass trajectories for different flight-vehicles. These software are built to run in the work station environment. OptiA system [5] is an interactive environment for the definition, solution, and analysis of nonlinear mathematical programming problems. OptiA solves optimal control problems using control parametrization (see, e.g., [11]). DIRCOL program [16] discretizes optimal control problems using direct collocation method and resulting finite dimensional problems are solved using sequential quadratic programming. DIRCOL utilizes variable discretization points and NPSOL subprogram [8]. OptiA and DIRCOL programs produce external files during the solution process which can be compiled with the suitable compiler both in PC and in workstation environments.

In the following we will first define the class of aircraft trajectory optimization problems whose solution process is to be automated. Then we will describe a convenient solution method for the automated optimization server. The description of VIATO software is given in section 4. The graphical user interface and the use of the software are demonstrated by example runs in section 5 and concluding remarks appear in section 6.

2. Aircraft trajectory optimization problems

In an optimal control problem the objective is to find an admissible control function that transforms a dynamical system from its initial state to another state so that a given objective or cost function is either minimized or maximized. In formulating an optimal control problem, one first has to construct a model of the dynamical system, for example, by identifying the state variables and specifying relations between them. After modeling of the system a suitable optimal control problem must be formulated. These tasks can be difficult and complex even for experts in mathematics and optimal control. It is obviously impossible to translate a real world problem into an optimal control problem without experience on mathematical modeling.

We avoid the formulation of optimal control problems by specifying the objectives of the problems and by fixing the aircraft equations of motion together with constraints for control and state variables in advance. This can be done in aircraft trajectory optimization tasks because structurally similar differential equations represent most point-mass models of aircraft.

The dynamics of an aircraft is described by a system of differential equations that constitute the state equation of the optimization problem. The equations of motion for the three-dimensional point-mass model of an aircraft are (see, e.g., [12]):

$$
\begin{aligned}
\dot{x} &= v\cos\gamma\cos\chi \\
\dot{y} &= v\cos\gamma\sin\chi \\
\dot{h} &= v\sin\gamma \\
\dot{v} &= \frac{1}{m}\{uT_{max}(h, M(h,v)) - (C_{D_0}(M(h,v)) + \\
&\quad C_{D_i}(M(h,v))(\frac{nmg}{S\frac{1}{2}\varrho(h)v^2} - a)^2)S\frac{1}{2}\varrho(h)v^2\} - g\sin\gamma \\
\dot{\gamma} &= \frac{g}{v}(n\cos\mu - \cos\gamma) \\
\dot{\chi} &= \frac{g\,n\sin\mu}{v\cos\gamma} \\
\dot{m} &= -cuT_{max}(h, M(h,v)).
\end{aligned}
$$

The state variables x, y, h, v, γ, χ and m refer to the x-range, the y-range, altitude, velocity, flight path angle, heading angle and mass of the aircraft, respectively. The normal acceleration of the aircraft is controlled with the loadfactor n and the tangential acceleration with the throttle setting $u\in[0,1]$. The loadfactor is the ratio of the lift force and the gravitational force that affect the aircraft. In three-dimensional flight the loadfactor can be directed away from the vertical plane with bank angle $\mu\in[-\pi, \pi]$. The gravitational acceleration g and the specific fuel consumption coefficient c are assumed to be constant. The aircraft mass is also included as a state variable because flight times of several hundred seconds may reduce the mass considerably. T_{max} denotes the maximum available thrust force, C_{D_0} and C_{D_i} denote zero-lift and induced drag coefficients, respectively. The Mach number M is a function of altitude and velocity, $\rho(h)$ denotes the air density and S refers to the reference wing area of the aircraft. The air density and the speed of the sound are taken from the ISA standard atmosphere. The modern flight control systems reduce the induced drag. In the point mass model the reduction is treated by using the positive constant a. The two dimensional equations of motion are similar to the equations above but the y-range, the heading angle and the bank angle are not present.

The pilot and the aircraft itself set some constraints for the state and the control variables. The loadfactor has lower and upper limits which depend on the flight altitude and the Mach number of the aircraft. The feasible region of flight is described by the minimum altitude, the minimum velocity and the maximum dynamic pressure constraints.

Different aircraft types can be distinguished from each other by a set of parameters which are included in the dynamic system or in the constraints. In VIATO software the aircraft is characterized by the wing area, the initial mass, the fuel consumption coefficient, the drag coefficients and the maximum thrust force and the bounds for loadfactor and maximum dynamic pressure.

At this stage VIATO is designed for minimum time problems where the objective function is the total flight time to the terminal state.

3. The solution method

The necessary conditions for optimality of the optimal control problem are given by the Pontryagin's Maximum Principle (see, e.g., [4]). The necessary conditions constitute a multipoint boundary value problem that can be solved using, e.g., multiple shooting methods (see, e.g., [14]). The methods that solve the optimal control problem using the necessary conditions are called indirect methods. These methods produce accurate results but they are not always useful for automated solution process because their convergence domain is small and thus the initial guess for the state and adjoint variables must be accurate enough. Furthermore, the switching structure of the solution must be known in advance, and the numerous integrations of state and adjoint equations require a lot of computational effort.

These difficulties can be overcome by replacing the original optimal control problem with a finite dimensional approximation. The approximation can be solved by using nonlinear programming. In this way larger convergence domain is achieved. Furthermore, one does not have to supply an estimate of the switching structure of the original problem because the state and the control constraints of the optimal control problem are treated as standard constraints of nonlinear optimization.

Several methods exist for converting optimal control problems into parameter optimization problems (for a review, see [11]). A suitable approach is to discretize both the control and the state variables. In this way integration of the state equations and the objective function is totally avoided. The state and control discretization can be carried out, for instance, using the direct collocation method or the scheme based on differential inclusions. The direct collocation method [10] seeks the solution of the problem from among piecewise defined polynomials that have to satisfy the state equations pointwisely. The scheme based on differential inclusions [15] replaces the differential equation constraint with a requirement that each state must lie in the set of attainability of the previous state. In practice, the set of attainability is formed by solving the control variables from the state equations. The capability of these two methods are compared in [13].

4. VIATO software

VIATO software is an interactive environment for solving optimization problems of flight. The software solves minimum time climb problems, minimum time trajectories to a fixed or a moving target on the vertical plane and three dimensional interception problems. A moving target is specified by its initial position, velocity and flight path angle.

VIATO has been implemented as a client-server application. It consists of a separate user interface (the client) and an optimization server. At this stage the user interface and the optimization program are run in the same computer. It is also possible to distribute the software such that the client and the server are run in separate computers and data exchange is carried out using a suitable network.

VIATO's graphical user interface was implemented with Delphi which is a component based application development environment for MS-Windows applications. The user interface displays optimal solutions graphically and maintains the aircraft model library.

In VIATO different aircraft types are characterized by a set of parameters which are stored in a model library. The values of the drag coefficients and the thrust force are known only at certain Mach numbers and at certain altitudes. The optimization routine needs these values at arbitrary states and so the data must be approximated using some continuous functions. The user interface creates automatically continuous and smooth rational polynomial or spline approximations for the values of the drag coefficients. The maximum thrust force is approximated by a two dimensional polynomial. The approximations are determined by the least-squares fitting.

The optimization server is an independent MS-Windows program that was implemented with FORTRAN. The server discretizes the problems using direct collocation or a scheme based on differential inclusions. As a default the software uses collocation method and six equidistant discretization points. The server solves the finite-dimensional approximation of the original optimal control problem using NPSOL program [8] which is a versatile implementation of sequential quadratic programming (SQP) (see, e.g., [6]). SQP is a method in which at each iteration step the objective function is approximated with a quadratic form and nonlinear constraints are linearized. The resulting quadratic problem is solved by a suitable quadratic programming method. Then a line search between the iteration point and the solution of the quadratic problem is carried out. SQP method converges rapidly and reliably when applied to a discretized optimal control problem (see, e.g. [9]).

In optimization tasks VIATO's optimization server chooses the initial guess for the decision variables from a straight line connecting the initial and the terminal conditions. A problem can also be solved using continuation with respect to the number of the discretization points. In this process the number of the discretization points increases after each solution and the previous solution is used to form the initial guess for the state and the control variables.

An essential part of the solution is the scaling of the variables and the constraints. Optimization routines generally assume that the decision variables and the constraints are roughly of the same magnitude. Badly scaled decision variables can lead to ill-conditioned objective function gradient and badly scaled constraints might generate near-singular Jacobian of the constraints. Because of these facts the variables and constraints with widely varying values may cause numerical errors for the iteration. A suitable form for scaling is to define new variables and constraints using a linear transformation [7]. For example, in VIATO each state and control variable is normalized by the absolute maximum of the initial guess.

The SQP-routine requires the values of the objective function and the constraints and their first derivatives at arbitrary points. The optimization server uses analytical derivatives for the objective function and for the constraints.

5. Example runs with VIATO

The software is demonstrated by solving two example aircraft trajectory optimization tasks. The example problems are a minimum time climb, where the total time needed in achieving a certain altitude and velocity is minimized, and a three-dimensional minimum time interception of a fixed target. These optimization tasks can be regarded as basic problems of military aviation. In addition, the example runs illustrate how VIATO can be used in sensitivity analysis. The software was run in a PC equipped with 66MHz 486-processor.

To run VIATO software, the user first has to choose the aircraft model from the model library. The continuous approximations for the drag coefficients and the drag forces are formed and displayed with the original data (see figure 1). The wing area and the mass of the aircraft are also shown. After the aircraft model has been fixed, the user can choose the problem type from four possible optimization tasks mentioned above.

In the examples we use realistic drag coefficients and maximum thrust force of a generic modern high-speed and agile aircraft.

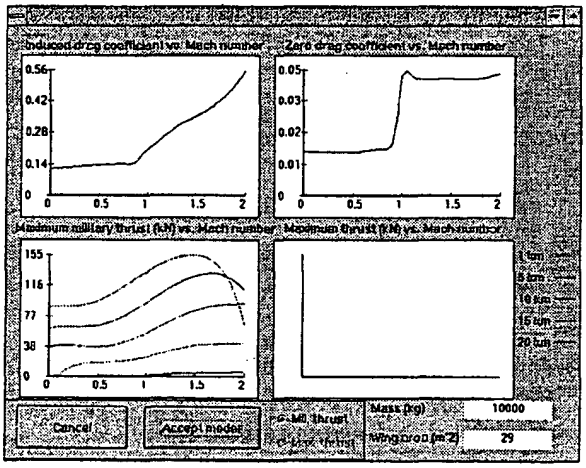

Figure 1: The aircraft model that is used in the example runs.

Minimum time climb on the vertical plane

When the problem type has been chosen, the initial and the terminal conditions are specified. The user can also choose the method of discretization, either direct collocation or differential inclusion, and the number of discretization points. In the example run the initial and the terminal conditions of the aircraft are:

$h(0) = 100$ m, $h(T) = 10000$ m,
$v(0) = 150$ m/s, $v(T) = 400$ m/s,
$\gamma(0) = 0.1$ rad,
$m(0) = 10000$ kg,

where T refers to the total flight time. The initial conditions correspond to flight conditions just after a take off.

Once the user interface has received the required information, it calls the optimization server which solves the problem and returns the optimal solution to the user interface. Optimal solutions are presented graphically and results can also be saved in files. Thus it is possible to read the solutions in other applications as well.

Solutions obtained with the two-dimensional point mass model are shown in two-dimensional plots, see figure 2. The optimal state and control variables can be plotted as a function of some other state or control variable, or the total flight time. The optimal trajectories can further be shown together with constant energy contours and contours of the time derivative of the aircraft's energy, also called Specific Excess Power, SEP, in the Mach number-altitude plane, see figure 3. In addition to the plots, the trajectories to a moving target can also be animated.

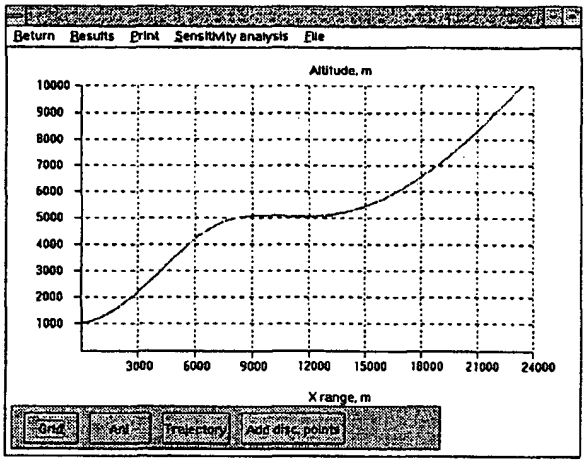

Figure 2: Visualization of two-dimensional trajectories in VIATO.

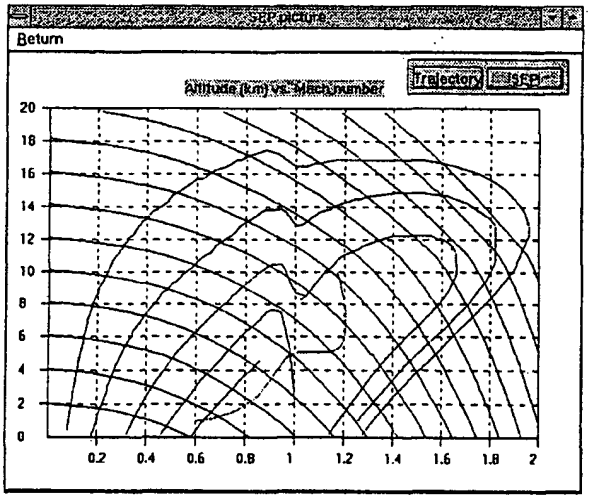

Figure 3: The optimal solution is shown with the SEP-plot.

At first the example minimum time climb problem was solved with 6 discretization points to get a rough initial estimate to the solution. Then the accuracy of the approximation was increased by adding the number of discretization points from 8 to 12 at two intervals. These solutions were produced such that the previous solution was used as the initial guess. In this way the accuracy of the solution was increased and the solution with 12 discretization points was obtained faster than solving the problem only once using 12 discretization points and a simple straight line initial guess.

The optimal trajectories of the example minimum time climb problem are shown in figure 4. The optimal flight time was 97 sec. for the problem with 6 discretization points and approximately 95 sec. for the problems with 8, 10 and 12 discretization points. The solution with 6 discretization points differs noticeably from the optimal solutions obtained with 8, 10 and 12 discretization points. The calculation time of the example problem grew from 30 sec. (6 discretization points) to 60 sec. (12 discretization points).

Three-dimensional interception

In the second example a three-dimensional interception problem was solved. The objective of the interception is to fly in minimum time from the initial state $x(0) = 0$ m, $y(0) = 0$ m, $h(0) = 5000$ m, $v(0) = 200$ m/s, $\gamma(0) = 0.1$ rad and $\chi(0) = 0.1$ rad to the terminal point $x(T) = 10000$ m, $y(T) = 15000$ m, $h(T) = 10000$ m, $\gamma(T) = 0$ rad and $\chi(T) = 0$ rad. T refers to the total flight time. The terminal velocity of the aircraft, $v(T)$, is free. The initial condition corresponds to a cruise situation.

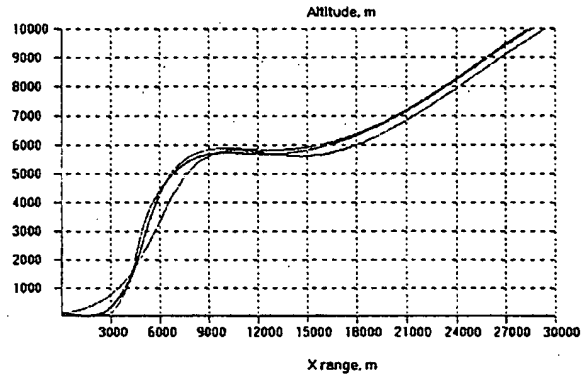

Figure 4: The minimum time climb trajectories from the take off conditions obtained with 6, 8, 10 and 12 discretization points.

In addition to two-dimensional plots, the optimal trajectory can now be plotted in a three-dimensional picture where the bank angle of the aircraft is also presented, see figure 5. The user can change the perspective, the size and the view angle of the three-dimensional picture. It is also possible to draw the projections of the optimal trajectory on x,y-, x,z- and y,z-planes.

The example three-dimensional interception problem was solved using 14 discretization points and direct collocation method. The three-dimensional time optimal trajectory is shown in figure 5. The interception took 63 sec. The dimension of the problem was larger and there were clearly more constraints than in the minimum time climb task. For that reason the calculation time increased to 310 sec.

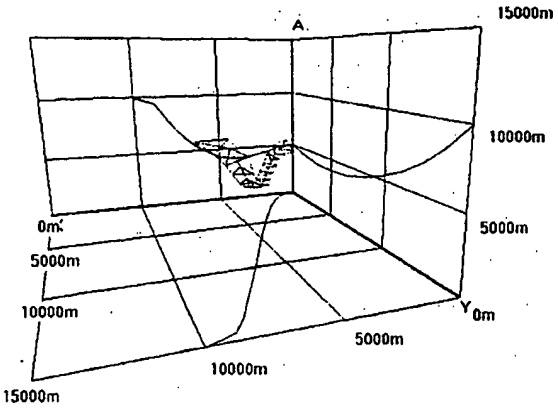

Figure 5: The time optimal trajectory to the fixed target in three dimensional space. The optimal flight time is 63 sec.

Sensitivity analysis

Solutions of optimal control problems depend on model parameters and therefore it is interesting to study the change of the optimal objective value and the optimal trajectories, as the value of a certain model parameter is varied. In VIATO software the varying parameter can be one of the initial or the terminal states, the initial mass, the maximum thrust force or the drag coefficients, for example.

The sensitivity analysis can be carried out after the problem has been solved once. At first the user must choose the varying parameter and determine its new value. Then the problem is solved again such that the solution obtained with the original value of the varying parameter is used as an initial guess for the decision variables. All solutions are visualized in the same way as the two example runs shown earlier.

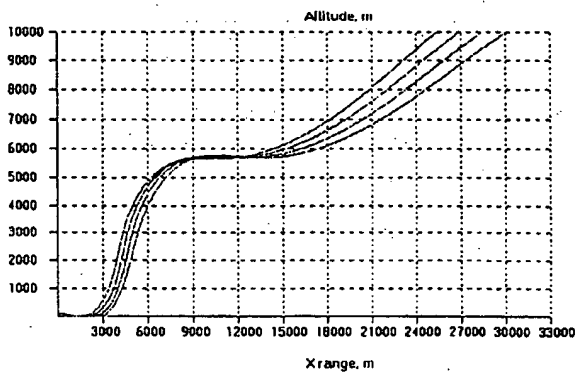

Figure 6: A family of minimum time climb trajectories obtained with 8 discretization points. The initial mass of the aircraft is changed in 500 kg intervals from 9000 kg to 10500 kg. The optimal flight times are 87, 91, 96 and 100 sec., respectively.

We have executed an example sensitivity analysis. The analysis deals with the previous minimum time climb problem. The sensitivity of the optimal control and state history is studied with respect to the initial mass of the aircraft.

The minimum time climb trajectory obtained with 8 discretization points in the first example run was used as a reference solution. Then the problem was solved four times such that the initial mass was changed in 500 kg intervals from 9000 kg to 10500 kg. The family of the optimal solutions is shown in figure 6. The percentual change of the optimal flight time (from 86 sec. to 100 sec.) and the initial mass were approximately equal. The calculation times varied between 30 and 50 sec.

6. Conclusion

We have introduced an approach towards automated solution of optimal flight trajectories. The approach was implemented in the interactive aircraft trajectory optimization software, VIATO. The structure of the aircraft models and the objective of the problem is specified in advance. Different aircraft types are distinguished by sets of parameters which are stored in a model library. VIATO provides a graphical user interface to facilitate interaction between the user and the optimization server. The user interface is run under MS-Windows operating system and it visualizes the optimal control and state histories produced by the optimization server graphically. The optimal control problems are solved reliably and in reasonable time using discretization and nonlinear programming. At this stage the software can be used to solve different climb and interception problems, where the objective is to minimize the total flight time. Sensitivity analysis which measures the impact of changing the model parameters on the optimal solution, can also be run easily. The example runs demonstrated that the optimal solutions are produced without difficulty and they are accurate enough for most purposes. Overall, the presented optimization environment proved a promising tool in solving complex optimal control problems automatically.

VIATO can be considered as a decision support tool for an efficient solution of decision-making and design problems in flight technics and tactics. In future it might be possible to apply the software to tasks like tactical training, allocation of recourses and technological validation.

References

[1] Betts J.T.: Issues in the Direct Transcription of Optimal Control Problems to Sparse Nonlinear Programs. Computational Optimal Control, Bulirsch R. and Kraft D., eds. Birkhäuser, 1994. (3-17)

[2] Bless R.R., Queen E.M., Cavanaugh M.D., Wetzel T.A. and Moerder D.D.: Variational Trajectory Optimization Tool Set, Technical Description and User's Manual. NASA Technical Memorandum 4442, 1993.

[3] Brauer G.L., Cornick D.E., Habeger A.R., Petersen F.M. and Stevenson R.: Program To Optimize Simulated Trajectories (POST). Volume I-Formulation Manual. NASA CR-132689, 1975.

[4] Bryson A.E. and Ho Y.C.: Applied Optimal Control, 2nd printing. Hemisphere publishing company, 1975.

[5] Dolezal J. and Fidler J.: Numerical Solution of Dynamic Optimization Problems Using Parametrization and OptiA Software. Applied Mathematics and Computation, Vol. 78, 1996. (111-121)

[6] Fletcher R.: Practical Methods of Optimization, 2nd Edition. John Wiley and Sons, 1987.

[7] Gill P.E. and Murray W.: Model Building and Practical Aspects of Nonlinear Programming. Computational Mathematical Programming, NATO ASI Series, Vol. F15, Schittkowski K., eds. Springer-Verlag, 1985. (209-247)

[8] Gill P.E., Murray W. and Wright M.H.: User's quide for NPSOL (Version 5.0). Report SOL 86-2. Department of Operation Research, Standford University, California, USA, 1986.

[9] Gill P.E., Murray W. and Saunders M.A.: Large-scale SQP Methods and Their Application in Trajectory Optimization. Computational Optimal Control, Bulirsch R. and Kraft D., eds. Birkhäuser, 1994. (29-42)

[10] Hargraves C.R. and Paris S.W.: Direct Trajectory Optimization Using Nonlinear Programming and Collocation. Journal of Guidance, Control and Dynamics. Vol 10, No. 4 (July-August), 1987. (338-342)

[11] Hull D.G.: Conversion of Optimal Control Problems into Parameter Optimization Problems. Journal of Guidance, Control and Dynamics. Vol. 20. No. 1 (January-February), 1997. (57-60)

[12] Miele A.: Flight Mechanics, vol 1. Addison-Wesley, 1962.

[13] Raivio T., Ehtamo H. and Hämäläinen R.P.: Aircraft Trajectory Optimization Using Nonlinear Programming. System Modeling and Optimization, Dolezal J. and Fidler J., eds. Chapman & Hall, 1996. (435-441)

[14] Roberts S.M. and Shipman J.S.: Two-Point Boundary Value Problems: Shooting Methods. Modern Analytical and Computational Methods in Science and Mathematics, No. 31. Elsevier, 1972.

[15] Seywald H.: Trajectory Optimization Based on Differential Inclusion. Journal of Guidance, Control and Dynamics. Vol 17, No. 3 (May-June), 1994. (480-487)

[16] Stryk O. von: Numerical Solution of Optimal Control Problems by Direct Collocation, in Bulirsch R., Miele A., Stoer J. and Well K.-H., eds., Optimal Control, International Series in Numerical Mathematics 111. Birkhäuser, Basel, 1993. (129-143)